

Sub  
B1

**Methods and Apparatus for Providing  
Manifold Array (ManArray)  
Program Context Switch with Array Reconfiguration Control**

**Related Applications**

The present application claims the benefit of U.S. Provisional Application Serial No. 60/140,244 entitled "Methods and Apparatus for Providing One-By-One Manifold Array (1x1 ManArray) Program Context Switch Control" and filed June 21, 1999 which is incorporated by reference herein in its entirety.

**Field of Invention**

The present invention relates generally to improvements in the manifold array (ManArray) architecture, and more particularly to advantageous methods and apparatus for providing efficient context switching between tasks in a ManArray processor environment, and advantageous methods and apparatus for array reconfiguration.

**Background of the Invention**

Members of the ManArray family of core processors are created by appropriately combining a number of basic building blocks. One of these building blocks is a unit that combines an array controller sequence processor (SP) with a processing element (PE). Another building block is a single PE. These building block elements are interconnected by the ManArray network and DMA subsystem to form different size array systems. By embedding an array operating mode bit, that controls the SP or PE execution, and communication instructions, that operate on the scalable high performance integrated interconnection network, in the

instruction set architecture, a scalable family of array cores, such as 1x1, 1x2, 2x2, 2x4, 4x4, and the like is produced. For example, a 1x1 ManArray core processor may suitably comprise a single set of execution units coupled with two independent compute register files. The processor's register files consist of a reconfigurable compute register file (CRF), providing either a 32x32-bit or 16x64-bit file configurations, an address register file (ARF) containing eight 32-bit registers and a set of status and control registers located in a miscellaneous register file (MRF) and special purpose registers (SPRs). The ManArray instruction set supports processor scalability in part through the use of an SP/PE bit (S/P-bit) contained in the ManArray instruction format. For array structures, this bit distinguishes whether the SP or the set of attached PEs will execute a particular instruction, though it is noted that some instructions actually are executed cooperatively by both the SP and PEs. By "execute an instruction", we mean that one or more processor registers or memories are updated based on the operation semantics.

In many applications, such as real time systems, multiple processes may have operating requirements with servicing deadlines that can only be met by sharing a processor on multiple independent tasks. Each task represents a context that is made up of the task's program, data, and machine state. To meet the deadlines imposed by the different processes, a real time operating system (OS) is typically used to manage when a task, from a set of multiple tasks, is to be executed on the processor. This real time OS can cause a context switch which may require the saving of the complete machine state for an existing context prior to loading the next context in the processor. Consequently, it is important to have a short context switching time in real time systems.

## Summary of the Invention

The merged SP/PE0 building block unit logically functions as a single context controller and by virtue of the merged PE provides supporting interfaces that allow additional PEs to be attached. In this single context controller environment, the S/P-bit is used to determine whether an instruction is to be executed in the SP only or is to be executed in the PE array. In one aspect of the present invention, the S/P-bit is used in a 1x1 array core to determine which register file, the SP's or the PE's, is to be accessed for each instruction execution. By treating the S/P-bit as a context-0/context-1 bit, the selection between two different register spaces effectively doubles the size of the register space for the SP. Thus, the 1x1 array core can be viewed as a single processor containing two register contexts that share a common set of execution units.

Note that this approach of using the S/P-bit for context switching purposes requires that for an instruction to access the PE register space, it must set the SP/PE bit in the instruction word to indicate it is a PE instruction. The implication of this requirement is that different forms of instructions are required to be used for accessing different registers. If it is desired to make use of both register files in a 1x1, for different contexts for example, the code must be explicitly targeted by using either PE or SP instructions. This limitation does not allow for seamless context switching between tasks since the task code is not uniform. As addressed further below, the present invention advantageously addresses these and other limitations providing improved context switch control.

Multiple register contexts are obtained in the ManArray processor by controlling how the array S/P-bit in the ManArray instruction format is used in conjunction with a context switch bit (CSB) for the context selection of the PE register file or the SP register file. In arrays consisting of more than a single PE, the software controllable context switch mechanism is used to

reconfigure the array to take advantage of the multiple context support the merged SP/PE provides. For example, a 1x1 can be configured as a 1x1 with context-0 and as a 1x0 with context-1, a 1x2 can be configured as a 1x2 with context-0 and as a 1x1 with context-1, and a 1x5 can be configured as a 1x5 with context-0 and as a 2x2 with context-1. Other array configurations are clearly possible using the present invention. In the 1x5/2x2 case, the two contexts could be a 1x5 with the sequential control context in the SP register files with context-0 and a 2x2 array context, where the sequential control context uses the PE0's register files with context-1.

These and other features, aspects and advantages of the invention will be apparent to those skilled in the art from the following detailed description taken together with the accompanying drawings.

#### **Brief Description of the Drawings**

Fig. 1 illustrates an exemplary 1x1 ManArray two context core operable in a first context as a 1x1 and in a second context as a 1x0 SP ManArray iVLIW processor in accordance with the present invention;

Fig. 2 provides a high-level view of the basic function of the S/P-bit and context switch bit (CSB) for improved context switch control in accordance with the present invention;

Fig. 3 specifies the logical operation of various array configurations for different settings of the CSB and the instruction's S/P-bit;

Fig. 4 illustrates an exemplary 1x2 two context ManArray processor configurable as a 1x2 in context-0 and as a 1x1 in context-1; and

Fig. 5 illustrates an exemplary 1x5 two context ManArray processor configurable as a 1x5 in context-0 and as a 2x2 in context-1.

## Detailed Description

Sub  
A2

Further details of a presently preferred ManArray core, architecture, and instructions for use in conjunction with the present invention are found in U.S. Patent Application Serial No. 08/885,310 filed June 30, 1997, now U.S. Patent No. 6,023,753, U.S. Patent Application Serial No. 08/949,122 filed October 10, 1997, U.S. Patent Application Serial No. 09/169,255 filed October 9, 1998, U.S. Patent Application Serial No. 09/169,256 filed October 9, 1998, U.S. Patent Application Serial No. 09/169,072 filed October 9, 1998, U.S. Patent Application Serial No. 09/187,539 filed November 6, 1998, U.S. Patent Application Serial No. 09/205,558 filed December 4, 1998, U.S. Patent Application Serial No. 09/215,081 filed December 18, 1998, U.S. Patent Application Serial No. 09/228,374 filed January 12, 1999 and entitled "Methods and Apparatus to Dynamically Reconfigure the Instruction Pipeline of an Indirect Very Long Instruction Word Scalable Processor", U.S. Patent Application Serial No. 09/238,446 filed January 28, 1999, U.S. Patent Application Serial No. 09/267,570 filed March 12, 1999, U.S. Patent Application Serial No. 09/337,839 filed June 22, 1999, U.S. Patent Application Serial No. 09/350,191 filed July 9, 1999, U.S. Patent Application Serial No. 09/422,015 filed October 21, 1999 entitled "Methods and Apparatus for Abbreviated Instruction and Configurable Processor Architecture", U.S. Patent Application Serial No. 09/432,705 filed November 2, 1999 entitled "Methods and Apparatus for Improved Motion Estimation for Video Encoding", U.S. Patent Application Serial No. 09/471,217 filed December 23, 1999 entitled "Methods and Apparatus for Providing Data Transfer Control", U.S. Patent Application Serial No. 09/472,372 filed December 23, 1999 entitled "Methods and Apparatus for Providing Direct Memory Access Control", U.S. Patent Application Serial No. \_\_\_\_\_ entitled "Methods and Apparatus for Data Dependent Address Operations and Efficient Variable Length Code Decoding in a VLIW Processor" filed

Cont  
A2

June 16, 2000, U.S. Patent Application Serial No. \_\_\_\_\_ entitled "Methods and Apparatus for Improved Efficiency in Pipeline Simulation and Emulation" filed June 21, 2000, U.S. Patent Application Serial No. \_\_\_\_\_ entitled "Methods and Apparatus for Initiating and Resynchronizing Multi-Cycle SIMD Instructions" filed June 21, 2000, U.S. Patent Application Serial No. \_\_\_\_\_ entitled "Methods and Apparatus for Generalized Event Detection and Action Specification in a Processor" filed June 21, 2000, and U.S. Patent Application Serial No. \_\_\_\_\_ entitled "Methods and Apparatus for Establishing Port Priority Functions in a VLIW Processor" filed June 21, 2000, as well as, Provisional Application Serial No. 60/113,637 entitled "Methods and Apparatus for Providing Direct Memory Access (DMA) Engine" filed December 23, 1998, Provisional Application Serial No. 60/113,555 entitled "Methods and Apparatus Providing Transfer Control" filed December 23, 1998, Provisional Application Serial No. 60/139,946 entitled "Methods and Apparatus for Data Dependent Address Operations and Efficient Variable Length Code Decoding in a VLIW Processor" filed June 18, 1999, Provisional Application Serial No. 60/140,245 entitled "Methods and Apparatus for Generalized Event Detection and Action Specification in a Processor" filed June 21, 1999, Provisional Application Serial No. 60/140,163 entitled "Methods and Apparatus for Improved Efficiency in Pipeline Simulation and Emulation" filed June 21, 1999, Provisional Application Serial No. 60/140,162 entitled "Methods and Apparatus for Initiating and Re-Synchronizing Multi-Cycle SIMD Instructions" filed June 21, 1999, Provisional Application Serial No. 60/140,244 entitled "Methods and Apparatus for Providing One-By-One Manifold Array (1x1 ManArray) Program Context Control" filed June 21, 1999, Provisional Application Serial No. 60/140,325 entitled "Methods and Apparatus for Establishing Port Priority Function in a VLIW Processor" filed June 21, 1999, Provisional Application Serial No. 60/140,425 entitled "Methods and Apparatus for

Cont  
A2

Parallel Processing Utilizing a Manifold Array (ManArray) Architecture and Instruction Syntax" filed June 22, 1999, Provisional Application Serial No. 60/165,337 entitled "Efficient Cosine Transform Implementations on the ManArray Architecture" filed November 12, 1999, and Provisional Application Serial No. 60/171,911 entitled "Methods and Apparatus for DMA Loading of Very Long Instruction Word Memory" filed December 23, 1999, Provisional Application Serial No. 60/184,668 entitled "Methods and Apparatus for Providing Bit-Reversal and Multicast Functions Utilizing DMA Controller" filed February 24, 2000, Provisional Application Serial No. 60/184,529 entitled "Methods and Apparatus for Scalable Array Processor Interrupt Detection and Response" filed February 24, 2000, Provisional Application Serial No. 60/184,560 entitled "Methods and Apparatus for Flexible Strength Coprocessing Interface" filed February 24, 2000, and Provisional Application Serial No. 60/203,629 entitled "Methods and Apparatus for Power Control in a Scalable Array of Processor Elements" filed May 12, 2000, respectively, all of which are assigned to the assignee of the present invention and incorporated by reference herein in their entirety.

In a presently preferred embodiment of the present invention, a ManArray 1x1 iVLIW single instruction multiple data stream (SIMD) processor 100 shown in Fig. 1 contains a controller sequence processor (SP) combined with processing element-0 (PE0) SP/PE0 101, as described in further detail in U.S. Application Serial No. 09/169,072 entitled "Methods and Apparatus for Dynamically Merging an Array Controller with an Array Processing Element".

The SP/PE0 101 contains a fetch controller 103 to allow the fetching of short instruction words (SIWs) from a B=32-bit instruction memory 105. The fetch controller 103 provides the typical functions needed in a programmable processor such as a program counter (PC), branch capability, digital signal processing, eventpoint (EP) loop operations, support for interrupts, and

also provides instruction memory management control which could include an instruction cache if needed by an application. In addition, the SIW I-Fetch controller 103 dispatches 32-bit SIWs to the other PEs that may be attached in an array, and PE0, in the case of the processor 100 of Fig. 1. The 32-bit SIWs are dispatched utilizing a 32-bit instruction bus 102.

Sub A3  
In this exemplary system 100, common elements are used throughout to simplify the explanation, though actual implementations are not so limited. For example, the execution units 131 in the combined SP/PE0 101 can be separated into a set of execution units optimized for the control function, e.g. fixed point execution units, and the PE0 as well as any other PE that could be attached can be optimized for a floating point application. For the purposes of this description, it is assumed that the execution units 131 are of the same type in the SP/PE0 and in the additional PE or PEs, such as PE1 of Fig. 4 or PEs 1, 2 or 3 of Fig. 5. In a similar manner, SP/PE0 and the other PE/PEs use a five instruction slot iVLIW architecture which contains a very long instruction word memory (VIM) memory 109 and an instruction decode and VIM controller function unit 107 which receives instructions as dispatched from the SP/PE0's I-Fetch unit 103 and generates the VIM addresses-and-control signals 108 required to access the iVLIWs stored in the VIM. These iVLIWs are identified by the letters SLAMD in VIM 109. The loading of the iVLIWs is described in further detail in U.S. Patent Application Serial No. 09/187,539 entitled "Methods and Apparatus for Efficient Synchronous MIMD Operations with iVLIW PE-to-PE Communication". Also contained in the SP/PE0 is a SP reconfigurable register file 111 and a PE reconfigurable register file 127 which is described in further detail in U.S. Patent Application Serial No. 09/169,255 entitled "Methods and Apparatus for Dynamic Instruction Controlled Reconfiguration Register File with Extended Precision".



Due to the combined nature of the SP/PE0 101, the data memory interface controller 125 must handle the data processing needs of both the SP controller, with SP data in memory 121, and PE0, with PE0 data in memory 123. The data memory interface controller 125 also provides a broadcast data bus interface (not shown in Fig. 1 as this Figure shows a single PE core) to attached PEs, special purpose registers (SPRs), and support for the ManArray eventpoint architecture. Any other PEs would contain their own physical data memory units, though the data stored in them is generally different as required by the local processing done on each PE. The local interface to these PE data memories is also a common design in any other attached PE. The interface to a host processor, other peripheral devices, and/or external memory can be done in many ways. The primary mechanism shown for completeness is contained in a direct memory access (DMA) control unit 181 that provides a scalable ManArray data bus 183 that connects to devices and interface units external to the ManArray core. The DMA control unit 181 provides the data flow and bus arbitration mechanisms needed for these external devices to interface to the ManArray core memories via the multiplexed bus interface represented by line 185. A high level view of a ManArray control bus (MCB) 191 is also shown.

All of the above noted patent applications are assigned to the assignee of the present invention and incorporated herein by reference in their entirety.

To provide for efficient context switching within a ManArray processor, a processor mode bit is provided in a control register in a miscellaneous register file (MRF). This bit is identified as a context switch bit (CSB). Fig. 2 illustrates a functional view of a system 200 for implementing the present invention. An S/P-bit and CSB bit control logic unit 202 contains the CSB and override logic. The control logic unit 202 provides enable signals 204 and 206 to multiplexers 208 and 210, respectively, to select where the result data from the execution units

212 are to be written. The result data is selectably written either to the SP configurable register file 214 or to the PE configurable register file 216. The control logic unit 202 also provides a select signal 218 to a multiplexer 220 to control which block of registers 214 or 216 that execution units 212 read data from. It is noted that in Fig. 2, the execution units 212 in the ManArray iVLIW processor may advantageously comprise five heterogeneous execution units which correspond to the five execution units 131 in Fig. 1. Also, the buses, multiplexers, and select control signals shown in Fig. 2 are indicated with multiple lines since in the ManArray processor such as shown in Fig. 1 there are eight 32-bit read ports and four 32-bit write ports for each 16x32-bit portion of both of the reconfigurable register files and each requires separate selection and control depending upon the instruction in execution and the machine state.

Specifically, the CSB bit in conjunction with the S/P-bit in PEO's control logic allows efficient context switching between tasks. Control specification 300 of Fig. 3 lists three exemplary array configurations and describes the register file use and array operating configuration for SP or PE instructions, as specified by the instruction's S/P-bit, depending upon the setting of the CSB bit. Table 310 indicates the ManArray architecture definition of the S/P-bit, which is present in the execution units' instruction formats. In general, other register files including the reconfigurable compute register files are shared between contexts. Specifically, in Fig. 3, the register files that are indicated to be shared are the address register file (ARF), the compute register file (CRF), and selected MRF and special purpose registers (SPRs) used by the execution units. The physical MxN column 304 indicates the physical array organization of PEs in the core processor, while the operating MxN column 312 depends upon the CSB value. It is noted that with the CSB bit set to zero, as seen in control specification entries 320, 322, 330, 332, 340, and 342, the SP operates in context-0 with SP instructions only executing in the SP on

SP resources and PE instructions only executing in any or all of the PEs on PE resources. With the CSB bit set to a one, as seen in control specification entries 324, 326, 334, 336, 344, and 346, the SP operates in context-1 which uses the PE0's register files. As described by this invention, each MxN core is a two context processor where one of the contexts uses SP-only resources for sequential control while the other context uses PE0's resources for sequential control.

By controlling the CSB-bit, an operating system (OS) can select a "context" for a task. In the 1x1 case, entries 320-326, where no PE instructions are used in a program, the core processor acts as a 1x0 with two contexts that the OS can freely assign as required by an application. In this 1x1 case, use of the CSB bit, rather than dependence on the S/P-bit only, allows the task code to be written in a uniform manner when using only the SP forms of instructions. Using PE instructions on PE0 even when the CSB bit is set to a 1, entry 326, is not likely an advantage, but can be optionally allowed effectively sharing PE0's context-1 register files between the SP and PE0.

The two other cases addressed herein, by way of example, namely a 1x2 and a 1x5, provide array reconfiguration dependent upon the context in operation. For example, in 1x2 system 400 of Fig. 4, the physical configuration of the processor is a merged SP/PE0 401 with an additional PE 451. With the CSB-bit set to a zero, inactive level, the core processor functions as a 1x2 as indicated by entries 330 and 332 in Fig. 3. When the CSB-bit is set active, then the SP takes over the use of PE0's register files 427, and other inferred files, as the second context. It is noted that for these configurations no SP instructions can be mixed with PE instructions in the physical PE0 since the register files are being used for program context switching purposes. When the SP is using PE0's register file resources as context-1, the additional PE is still available for use. Consequently, the operating configuration switches from a 1x2 to a 1x1 in the

second context. To allow the array, the single additional PE in this case, to function properly in PE identity (ID) dependent operations, such as for control of cluster switch 471, the additional PE switches to a virtual identity as PE0 when the CSB-bit is active. For example, the SPRECV instructions specify which PE is to send a source register to the cluster switch and identify in the SP/PE0 from which PE# the data is to be received from. For code written for a 1x1, the SPRECV instruction, if used, would specify PE0 to receive data from. For this operation to happen correctly in the physical 1x2 reconfigured as a 1x1, the physical PE1 switches to a virtual identity of PE0 and responds to the SPRECV 1x1\_PE0 instruction.

This approach may also be used on larger arrays, such as 1x5 ManArray processor 500 shown in Fig. 5 having four additional PEs 551, 553, 555, and 557 in addition to PE0 which is part of SP/PE0 501. Fig. 5 uses the following notation for the PEs: PE virtual ID/physical ID. In processor 500 of Fig. 5, there are five physical PEs which operate as a 1x5 with the SP using the SP register files 511 as context-0 when the CSB bit is inactive. When the CSB bit is active, the PE array reconfigures itself into a 2x2 with the SP taking over PE0's register files 527, and other inferred files, for context-1. Each of the PEs switches to a virtual identity such that code written for a 2x2 PE array functions correctly on the reconfigured organization. Each PE supports the decode function for the two identified PEs. For example, PE 2/3 555 responds as PE3, its physical ID, when the CSB bit is inactive and responds as PE2, its virtual ID, when the CSB bit is active. The concepts of virtual PEs and cluster switch control is covered in further detail in U.S. Application Serial No. 09/169,256 entitled "Methods and Apparatus for ManArray PE-PE Switch Control". Note that the cluster switch is extended to support five PEs in Fig. 5 which is allowed by the general form of the ManArray interconnection network and covered in additional detail in U.S. Patent No. 6,023,753 entitled "Manifold Array Processor" and U.S. Application

Serial No. 08/949,122 entitled "Methods and Apparatus for Manifold Array Processing", both of which are incorporated herein by reference in their entirety.

It is noted that when the SP uses the PE0 resources as specified by a context switch, some SP registers can remain SP-only regardless of the setting of the CSB to minimize implementation costs. These register addresses map to resources which are shared between any context such as interrupt control/status registers, cycle count registers, mode control registers, etc.

Sub  
A4

To further support the context switch mechanism and provide support for multiple contexts, an additional mechanism is added to allow one of the register files to be saved and restored from memory in the background while a task is using another register file referred to as the foreground register file. One mechanism used for this takes advantage of unused load and store unit instruction slots to perform this context switch save and restore operation. Essentially, "background" store and load instructions, together with a means of indexing through a register file, are activated whenever a task is not executing a foreground load or store instruction. A pair of background address registers is required to provide the store and load addresses for the register context switch. The "background" store and load instructions are pre-stored context switch save and restore instructions which, when enabled, operate in the background until the save and restore operation has completed. Use of the eventpoint architecture is one mechanism that can be set up to test for the lack of foreground store and load instruction execution and trigger a background store and load instruction to execute. Suitable eventpoint architecture is covered in more detail in U.S. Provisional Application Serial No. 60/140,245 entitled "Methods and Apparatus for Generalized Event Detection and Action Specification in a Processor" and U.S. Application Serial No. \_\_\_\_\_ having the same title and filed June 21, 2000, both of which are incorporated by reference herein in their entirety. A status bit is also used to indicate

Cont  
A4

the progress of the context switch so that, if preempted, it could be allowed to complete before another program context was initiated. Further details of a presently preferred register file indexing mechanism are provided in U.S. Patent Application Serial No. 09/267,570 entitled "Register File Indexing Methods and Apparatus for Providing Indirect Control of Register Addressing in a VLIW Processor" filed March 12, 1999 and incorporated by reference herein in its entirety. This register file indexing mechanism is preferably used for register file access.

Using the above background save and restore mechanisms, an OS could support two task context in registers at any given time and provide the ability to switch contexts in one set of registers while executing from the other. The register-based task contexts would allow for very low-overhead context switching.

While the present invention is disclosed in the context of a presently preferred embodiment, it will be recognized that a wide variety of implementations may be employed by persons of ordinary skill in the art consistent with the above discussion and the claims which follow below.